Fast Mapping on Myrinet Networks

Erik A. Hendriks
Advanced Computing Laboratory
Los Alamos National Laboratory*
hendriks@lanl.gov

March 17, 2004

Abstract

This paper presents an alternate method for discovering new switches and comparing switches when mapping a Myrinet network. Existing methods for mapping Myrinet networks rely on timeouts and are prone to false negatives due to network deadlock. Our algorithm increases the mapper's speed by providing a fast negative answer without relying on a timeout. In addition, the mapper's reliability is increased because it allows the mapper to detect if network deadlock has occurred. Mapping time on our 1024 port network has been reduced from approximately 7 minutes to 15 seconds.

1 Introduction

Myrinet is a source routed network. Each node has a table of routes to reach the other nodes in the network. Before routes can be generated a *mapper* program needs to produce a map of the network. The mapper runs on one of the nodes in the network and generates a network map by sending a series of *scout* packets into the network. These scouts probe for network features such as switches and hosts. The mapping step can be very time consuming since thousands of scouts are required to map even moderately sized networks.

Being able to produce a network map quickly has a direct impact on the performance of large systems. On Science Appliance systems [2] like Ed, Pink (1024 nodes) and ASCI Lightning (1408 nodes) where the Myrinet network is used for all system management as well as by applications, the network must be configured and working before moving on to any other step in cluster setup.

In the case of network failures, the mapper speed impacts how fast the system can react and map around failures. Isolation of network problems on a large network almost always involves some degree of trial and error and the mapper must remap the network any time a change is made.

In this paper we present the guarded scouting algorithm for scouting a Myrinet network. The guarded scouting algorithm has decreased the the time required for mapping by between one and two orders of magnitude depending on the version of the Myricom mapper that was used and the size of the network.

2 Background

This section describes how routing and mapping on a Myrinet network. The first few bytes at the beginning of every Myrinet packet is the route [1]. A route is a series of bytes. Each byte is an offset in a switch. For example, if a packet comes in on port 3 and the next routing byte is -2 the packet will leave the switch on port 1. Each switch that a packet goes through will strip off the first byte and route the packet accordingly. A routing byte of 0 is legal — it will route the packet back out the same port it came in on. The routes used to reached every node in the network are computed using a network map and stored in a table of routes in every node.

[*] Los Alamos National Laboratory is operated by the University of California for the National Nuclear Security Administration of the United States Department of Energy under contract W-7405-ENG-36. LANL LA-UR-04-1655

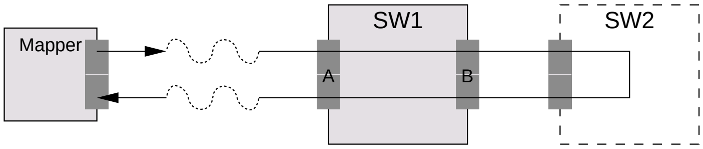

Figure 1: **Switch Scouting.** This figure shows the path of the scout packet (S) in the case where the mapper is probing for the existence of a switch. The mapper knows that SW_1 exists and is probing for the existence of SW_2. If SW_2 exists, the scout packet will return to the mapper. Otherwise, the mapper will timeout and conclude that SW_2 doesn't exist.

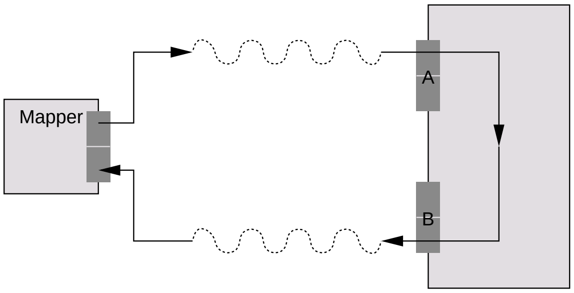

Figure 2: **Switch Comparison.** If the same switch is reached by two different routes, the mapper can verify this by sending a scout to the switch using one route and back with the other. In this example the mapper has a route to ports A and B on this switch. The packet is sent to port A, it makes one hop inside the switch to port B and returns using the reverse of the route to port B.

The mapper builds a map of the network by sending *scout* packets. The scout packets are small packets sent out with particular routes. Some of these will end up returning to the mapper and some will not. The mapper uses scouts to answer three different questions:

1. Given a route, is there a host there?
2. Given a route, is there a switch there?
3. Given two routes to switches, do they lead to different ports on the same switch?

Probing for a host is relatively simple. The host will respond to the scout with a packet identifying itself. Each host is uniquely identified by a MAC address.

Probing for Myrinet switches is somewhat more complicated. Switches do not identify themselves in any way. The only way the mapper can detect the presence of a switch is by sending a packet through it. Figure 1 illustrates the procedure used by the mapper to detect a switch. The mapper knows that the switch SW_1 exists and it has a route to port A on SW_1. The mapper is probing for the existence of the switch SW_2 on port B of SW_1. The mapper tries to send a scout packet too SW_2 and have it loop back to the mapper. If the scout packet returns to the mapper, then SW_2 exists. Otherwise it doesn't.

This only tests for the *existence* of a new switch. It does not identify the switch. There are usually many routes to each switch and it is possible that the mapper has already seen this switch using some other route. This leads to the need to answer question 3: Given two routes to switches, do they lead to different ports on the same switch?

This question is answered in a similar fashion. In this case the mapper has two different routes to two switches that it wants to compare to see if they are the same switch. The mapper can determine if these two switches are the same switch by sending a packet which will go to the switch using one route and return from the switch using the other route. Figure 2 illustrates the switch comparison procedure. In this example, the mapper knows the routes to two switch ports (A and B). The scout packet will take the route to port A, make a hop in the switch and use the route to B (reversed) to get back to the mapper.

If the scout makes it back to the mapper, the mapper knows that both of those routes lead to the same switch. Since the mapper supplied the hop to be taken in the switch, this also tells the mapper the relative numbering of the two ports.

By answering these three questions over and over again the mapper can build a map of the network. When mapping a large network (*e.g.* 1024 hosts) the bulk of the time is spent doing switch comparison. There are several practical problems:

Timeouts: Negative answers are obtained by timeouts in all three cases. The timeout must be long enough to handle the worst case. These timeouts are *much* longer than the time required for a positive response which is the time required for a packet to return to the sender. The answer is negative more often than positive so the mapper spends the bulk of its time waiting on timeouts.

Deadlock: The routes required to map the network are not deadlock safe. If there is other traffic on the network — which is almost always the case — the scout packets can cause deadlock with other traffic on the network. The switches solve the deadlock problem by simply dropping packets if there is no progress on a switch port for a long period of time ($\sim$ 1s). This is disturbing to any applications which may be running and problematic for the mapper because it leads to false negatives. The deadlock problem is unavoidable in general. Avoiding deadlock requires knowledge of the network topology.

This paper presents the guarded scouting algorithm. The guarded scouting algorithm provides an improved method for answering questions 2 and 3. It also provides a fast *no* answer as well as a fast *yes* answer. It also allows the mapper to detect that deadlock as occurred in the network. This makes it possible for the mapper to avoid false negatives as well allowing it to back-off and allow the network to clear itself when a deadlock occurs. Detecting deadlock virtually eliminates false negatives.

The guarded scouting algorithm sends additional scout packets (which we will call *guard* packets) immediately after the network scouts. Guard packets are sent along known good routes. They are sent after the scout packets in such a way that they are guaranteed to arrive after the scout packets. This allows the mapper to obtain a *no* response quickly if the guard returns and the scout has not returned. Also, if the guard packet goes missing, it is likely that a deadlock has occurred.

3 Presumptions

These are the presumptions that were made in designing the guarded scouting algorithm. These assumptions appear to pan out pretty well in practice.

- Raw sends are sent in order. This statement is stronger than necessary since the mapper can wait for one send to complete before starting the next one. The real presumption is that the MCP will not reorder receives for us after the fact.
- Raw sends are received in the order. The user space application must receive receive events in the order that the packets are received on the network.
- The network links are FIFOs. There is no way for one packet to pass another packet on the network if they both take the same route.
- The network is at least mostly reliable. The network does not drop packets for no reason. The error rate is low. A small number of retries should be good enough to deal with CRC errors and the like.

4 Scouting for New Switches

Figure 3 illustrates scouting for a new switch using guard packets. The scout packet (S) takes the same route as it does in the example without guard packets (Figure 1). The guard packet (G) is sent immediately after S and loops back inside of SW_1 instead of SW_2. Since the mapper knows that SW_1 exists, the route that G will take is known to be good.

The idea behind using the guard scout is that S will stay ahead of G. Then G's arrival will indicate that S is lost. The mapper's conclusion is based on which packet (S or G) returns first:

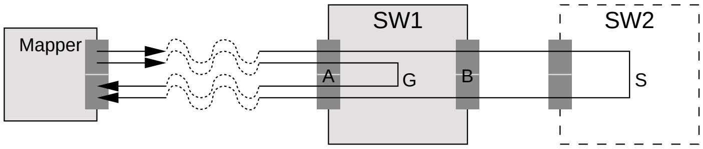

Figure 3: **Switch Scouting with Guard Packets.** This figure shows the paths of the scout packet (S) and the guard packet (G) in the case where the mapper is probing for the existence of a switch. The mapper knows that SW_1 exists and is probing for the existence of SW_2. The guard packet is sent after the scout packet. The scout should be long enough to occupy the input and output of port A on SW_1. This prevents the guard packet from passing the scout.

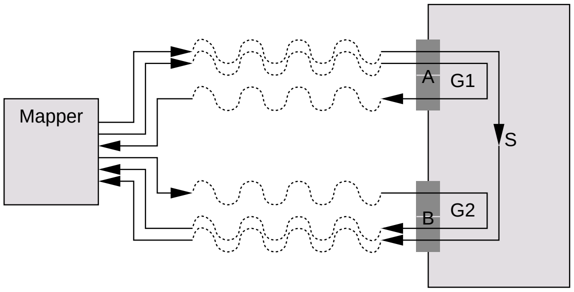

Figure 4: **Switch Comparison.** This figure shows the paths of the scout packet (S) and the guard packets for switch comparison. The mapper has two routes to two different switch ports (1 and 2). It wants to determine if these two ports are on the same switch. This diagram shows the routes of the scout and guard packets in the case that they actually are the same switch.

Packet S Returns First

If S returns to the mapper, then the mapper can conclude that the switch SW_2 exists. This case is the same as if there were no guard packets.

Packet G Returns First

If G returns to the mapper first, then the mapper concludes that S is lost since G can't get ahead of S. This implies that the switch SW_2 doesn't exist. This gives the mapper a fast *no* response without a timeout.

Neither returns in a short time

Since G's route was known to be good, the mapper concludes that some kind of network problem has occurred — most likely a network deadlock. The mapper waits and retries.

In order for S to stay ahead of G it must be long enough to occupy the input half of port A on SW_1 as well as the output half at the same time. The cut-through nature of Myrinet switches should make this possible. This will prevent G from getting through SW_1 before S.

There is a chance that there is a deadlock or some other problem (*e.g.* CRC error) that affects S but not G. This could cause G to come back first even though the switch exists. A small number of retries should be sufficient for avoiding this case.

5 Comparing Switches

A technique similar to the one used for switch scouting can be used for switch comparison. Figure 4 illustrates switch comparison with guard packets. Two guard packets are required in this case — one to follow the scout packet to the switch and one to follow the scout packet back.

The mapper has two known good routes that lead to switches in the network. The mapper is checking to see if both of those routes lead to different ports on the same switch. The scout packet uses one route to get to the switch and tries to use the other route to get back to the mapper node. This is the same route that the scout packet takes in the example without guard packets (Figure 2).

The first guard packet (G_1) is sent immediately following the scout packet (S). G_1 follows S along the route to the switch and then returns the way it came. Since G_1 follows S, G_1's return indicates that S has made it through the switch. After G_1 returns, the mapper sends G_2 along the other path to the switch — the path that S will use to return to the mapper. Since S has already passed through the switch, G_2 will leave through port B after S and follow S back from the switch. The mapper's conclusion is based on which packet (S or G) returns first:

Packet S Returns First

If S returns to the mapper, then the mapper can conclude that the two routes lead to same switch. This case is the same as if there were no guard packets. If S returns before G_1 there is no need to send G_2.

Packet G_2 Returns First

If G_2 returns to the mapper first, then the mapper concludes that S is lost since G_2 can't get ahead of S. This implies that the two routes do not lead to the same switch. This gives the mapper a fast *no* response without a timeout.

Nothing returns in a short time

Since G_1's and G_2's routes were known to be good, the mapper concludes that some kind of network problem has occurred — most likely a network deadlock. The mapper will wait and retry.

There is a possibility that S would be dropped due to some network problem (*e.g.* CRC error). G_2 could return first in that case. A small number of retries should be sufficient for avoiding this case.

6 Conclusion

Removing the reliance on timeouts has dramatically increased the speed of network mapping on our large clusters. Our test system is *Pink* which is a 1024 node cluster with a 1024 port Myrinet network. This network has 320 16-port switches in it. Our improved mapper reduced the mapping time from approximately 7 minutes to 15 seconds with a quiet network. In cases where there is a fair amount of traffic on the network, our mapper has proven to be more resilient since it has a good chance of detecting when a deadlock has occurred in the network.

Availability. The mapper software described in this paper is freely available on the web. It is licensed under the terms of the GPL.
`http://www.clustermatic.org/download/gm_route-1.0.tar.gz`

References

[1] Myrinet-on-VME protocol specification draft standard. Technical report, August 1998.

[2] Sung-Eun Choi, Erik A. Hendriks, Aaron J. Marks, Ronald G. Minnich, and Matthew J. Sottile. Life with Ed: A case study of a LinuxBIOS/BProc cluster. In *Proceedings of the 16th Annual International Symposium on High Performance Computing Systems and Applications*, June 2002.